REMARKS

Applicant wishes to thank the Examiner for the attention accorded to the instant application, and respectfully requests reconsideration of the application as amended.

Formal Matters

Claims 76, 77, 79 and 81-98 are currently pending in the application. Claims 78 and 80 are canceled; claims 1-75 have previously been canceled. Independent claims 76 and 98 are amended to recite that the identification of the one or more instructions and/or one or more variables is based on information obtained from an initial or an intermediate state of the creation process resulting in the executable file. Support for this amendment can be found in original claim 78. Claims 79 and 81 are amended to depend from claim 76 instead of canceled claim 78.

Response To Rejections Under 35 U.S.C.§ 103

The Examiner rejected claims 76-98 under 35 U.S.C. § 103(a) as unpatentable over Krishnan et al., U.S. Patent 6,141,698 (hereinafter "Krishnan") in view of Horning et al., U.S. Patent Application Publication No. 2005/0183072 (hereinafter "Horning"). This rejection should be withdrawn based on the comments and remarks herein.

The Examiner contends that Krishnan discloses considering information obtained from an initial or an intermediate state of the creation process resulting in the executable file. Applicant respectfully disagrees.

Krishnan augments the functionality of an application by considering information regarding the application that is contained only in the application's executable file. No other sources of information regarding the application are taken into account. In particular, no source code or object code, that is, information from an initial stage or intermediate state of the creation process, is considered by Krishnan.

Applicant points out that the process of creating an executable file from its source code representation consists of three stages. First, a compiler translates the set of source code files in a high-level language into a functionally equivalent set of assembly language source code files. Second, an assembler translates the set of assembly language source code files into a functionally equivalent set of object files. Third, a linker translates the set of object files together with object files from third party libraries into a single executable file.

It is important to observe that the transition from each stage to the next involves a loss of information. For example, in contrast to source code in a high-level language, assembly language source code files do not include type information for variables. Object files, as another example, do not contain information for symbols that are not externally visible, i.e., not required for linking. Executable files, as yet another example, do not contain symbol information at all.

Hence, considering information present in any of these stages leads to richer knowledge about an application than considering only information contained in the application's executable file. This superior knowledge about an application is what enables applicant's inventive method to apply more complex transformations to an executable file than Krishnan. Accordingly, the independent claims of the present application recite that "the identification of the one or more instructions and/or one or more variables is based on information obtained from an initial or an intermediate state of the creation process resulting in the executable file" so that the present inventive technique employs information from source code (an initial state) or compiled/assembled/ object code (an intermediate state).

In contrast, Krishnan teaches or suggests self-contained transformations to an application's executable file designed to operate exclusively on the executable file, as opposed to transformations that consider other sources of information regarding the application. Krishnan points out that the person modifying an executable file "may not have access to the source code" (column 1, lines 24-25). In particular, Krishnan teaches "two methods for injecting a DLL into existing executable code", an import table and DLL loader code, along with the injection of the security code (column 2, lines 57-65). Krishnan also teaches an incremental encryption (column 2, lines 66-67). All of these transformations exclusively consider only information contained in the application's executable file for modifying the same.

The first of Krishnan's two methods for injecting a DLL into existing executable code is the Import Table Method. It is described with reference to Fig. 5. As can be seen, the import table method exclusively operates on information contained in the executable file to be modified. The method includes the following steps, and file relationships.

501 ("Determine executable file type") – This relates to the executable file.

502 ("Known executable type?") – This also relates to the executable file.

503 ("Locate application import table") – This relates to the import table, which is part of the executable file.

504 ("Add DLL entry to import table") – This relates to the import table, which is part of the executable file.

505 ("Add stub function entry to DLL entry") – This relates to the entry to be added to the import table, which is part of the executable file.

506 ("Adjust relocatable addresses") – This relates to addresses within the executable file, which have to be corrected as the newly inserted entry increases the size of the import table, which is part of the executable file.

The second of Krishnan's two methods for injecting a DLL into existing executable code is the DLL Loader Code method. It is described with reference to Fig. 7. As can be seen, the DLL loader code method exclusively operates on information contained in the executable file to be modified. This method includes the following steps, and file relationships.

701 ("Determine executable file type") - This relates to the executable file.

702 ("Known executable type?") - This relates to the executable file.

703 ("Locate executable code in file") -This relates to the executable file.

704 ("Save application entry point") - This relates to the application entry point, which is part of the executable file.

705 ("Add DLL loader code to located code") -This relates to code contained in the file of 703, which is the executable file. New code is inserted into the code of 703.

706 ("application entrypoint = address of added DLL loader code") - This relates to the application entry point and the added DLL loader code, both of which are part of the executable file.

707 ("Adjust relocatable addresses") - This relates to addresses in the executable file that have to be adjusted because previously existing code in the executable file has to be moved to make room for the DLL loader code.

The injection of security code method, used by both the import table and the DLL loader code methods, is described with reference to Fig. 9. As can be seen, this injection of security code method exclusively operates on information contained in the executable file to be modified. This method includes the following steps, and file relationships.

901 ("Perform and save checksums; save application entry point") -This relates to checksums over parts of the application. As checksums have to be verified at runtime and at runtime the only available representation of the application is its executable file, the checksums must be computed over parts of the executable file. This also relates to the application entry point, which is part of the executable file.

902 ("Encrypt checksums and application entry point and store in a determined location") – This relates to a determined location in the application. As the information has to be accessed at runtime and at runtime the only available representation of the application is its executable file, this determined location must be in the executable file. This also relates to the application entry point, which is part of the executable file.

903 ("Incremental_Encrypt portions of application code") - This relates to incremental encryption, which is discussed in more detail below.

904 ("Locate executable code in file") -As the process of Fig. 9 is invoked at the end of the processes of Fig. 5 and Fig. 7, this relates to the executable files of Fig. 5 and Fig. 7.

905 ("Add checksum verification code to located code") - This relates to the code of 904, which is code in the executable file. New code is inserted into the code of 904.

909 ("Add other security checks to located code") – This relates to the code of 904, which is code in the executable file. New code is inserted into the code of 904.

906 ("Add Incremental_Decrypt code to located code") -This also relates to the code of 904, which is code in the executable file. New code is inserted into the code of 904.

907 ("Add code to decrypt and jump to saved application entry point") -This also relates to the code of 904, which is code in the executable file. Moreover, it relates to the application entry point, which is part of the executable file. New code is inserted into the code of 904.

908 ("Adjust relocatable addresses") -This relates to addresses in the executable file that have to be adjusted because previously existing code in the executable file has to be moved to make room for the checksum verification code, the Incremental_Decrypt code, and code to decrypt and jump to the saved application entry point.

Krishnan's Incremental Encryption scheme operates on blocks, which are determined by the method described with reference to Fig. 11. This "routine first scans the non-encrypted execution code and adjusts the portion of the code to be encrypted" (column 13, lines 47-49). Decryption of the code happens at runtime, so the code that is decrypted must be the code contained in the executable file, as the executable file is the only representation of the application available at runtime. Moreover, decryption must be applied to the same code to which encryption was applied. Accordingly, the code to which encryption is applied is also the code contained in the executable file. Hence, the above quote relates to the code of the executable file. This states that the algorithm of Fig. 11 operates on the code of the executable file.

Encryption of the individual blocks is described with reference to Fig. 10. As the encrypted blocks of code are decrypted at runtime, Fig. 10 must relate to code in the executable file, because the executable file is the only representation of the application available at runtime. Thus, as discussed above, incremental encryption exclusively operates on information contained in the executable file to be modified.

Therefore, Krishnan does not teach or suggest operating on information other than that in the executable file to be modified. Horning does not overcome this deficiency and the Examiner does not state otherwise. Thus, the present application advantageously provides more knowledge regarding an application so that the modification of the application's executable file is more secure and the rights of the software supplier and the rights of all subjects involved in the distribution process are protected.

It has been held by the courts that to establish *prima facie* obviousness of a claimed invention, all the claim limitations must be taught or suggested by the prior art. See, *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). As illustrated above, the hypothetical combination of Krishnan and Horning does not teach or suggest the identification of the one or more instructions and/or one or more variables is based on information obtained from an initial or an intermediate state of the creation process resulting in the executable file, and does not disclose or suggest each claim limitation of independent claims 76 and 98. Hence, *prima facie* obviousness is not established, so that claims 76 and 98 are patentably distinguishable over the art of record in the application. Claims 77, 79, and 81-97 depend from claim 76, each dependent claim incorporating all of the claim limitations of the base claim. Thus, these dependent claims are not anticipated by the art of record in the application for at least the reasons that the base claim is not anticipated by the art of record in the application. Claims 78 and 80 are canceled. Accordingly, this rejection should be withdrawn.

Conclusion

In light of the foregoing, Applicant respectfully submits that all pending claims recite patentable subject matter, and kindly solicits an early and favorable indication of allowability. If the Examiner has any reservation in allowing the claims, and believes a telephone interview would advance prosecution, he is kindly requested to telephone the undersigned at his earliest convenience.

Respectfully submitted,

[illegible]

Katherine R. Vieyra
Registration No. 47,155

SCULLY, SCOTT, MURPHY & PRESSER, P.C.
400 Garden City Plaza, Suite 300
Garden City, New York 11530
(516) 742-4343

KRV:jam